Journal of Computer and System Sciences 76 (2010) 593–608

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Comparing trees via crossing minimization

Henning Fernau [a,*], Michael Kaufmann [b], Mathias Poths [b]

[a] Universität Trier, FB IV—Abteilung Informatik, 54286 Trier, Germany
[b] Universität Tübingen, WSI für Informatik, Sand 13, 72076 Tübingen, Germany

ARTICLE INFO

Article history:
Received 5 March 2008
Received in revised form 22 October 2009
Available online 28 October 2009

Keywords:
Drawing tanglegrams
Crossing minimization
Parameterized algorithms

ABSTRACT

We consider the following problem (and variants thereof) that has important applications in the construction and evaluation of phylogenetic trees: Two rooted unordered binary trees with the same number of leaves have to be embedded in two layers in the plane such that the leaves are aligned in two adjacent layers. Additional matching edges between the leaves give a one-to-one correspondence between pairs of leaves of the different trees. Our goal is to find two planar embeddings of the two trees (drawn without crossings) that minimize the number of crossings of the matching edges. We derive both (classical) complexity results and (parameterized) algorithms for this problem (and some variants thereof). [1]

1. Introduction

Motivation. The comparison of trees has found applications in various areas like text data bases, bioinformatics, compiler construction etc. Various algorithms for different models for comparisons have been proposed. Keywords are tree matching, tree alignment, tree editing [25–27,32,37], a nice overview being [6]. Motivated by discussions with researchers from bioinformatics like D. Huson and D. Bryant, we started our study on comparing two binary tree structures by finding two most similar left-to-right orderings of their leaves. The ordering of the leaves can be made best visible by drawing one tree downward from the root and aligning all the leaves on a horizontal line. The leaves of the other tree are also aligned parallel to the first line and the corresponding tree structure is drawn from the leaves upside downward to the root. Corresponding leaves from the two trees are connected by matching edges. The problem directly arises in the comparison of phylogenetic trees, see the book edited by R.D.M. Page [31]. In this context, the drawing of two (phylogenetic) trees (as described above) is also called a *tanglegram*. Related problems have been considered in a research on phylogenetic trees for host-parasites pairs [9,33]. M.A. Charleston also provides an experimental environment for comparing phylogenetic trees with a software package called TREEMAP, version 3, see `http://www.it.usyd.edu.au/~mcharles/`, based on earlier versions developed by R.D.M. Page, where also an online description of TREEMAP (version 1) can be found, see `http://taxonomy.zoology.gla.ac.uk/rod/treemap/ch3.html`. The quoted Chapter 3 also contains a drawing of a tanglegram. It says in the documentation: "You can alter the appearance of the trees by … rotating descendants about their ancestor, thus "untangling" the trees. To do this, click on the desired branch. With some practice, you'll quickly be able to obtain a pleasing display. Note that you're not actually altering the phylogenetic relationships specified by the trees,

* Corresponding author.
E-mail addresses: fernau@uni-trier.de (H. Fernau), mk@informatik.uni-tuebingen.de (M. Kaufmann), poths@informatik.uni-tuebingen.de (M. Poths).

[1] A preliminary version of this paper has been presented at the FST&TCS conference in 2005. Most of the work of the first author has been made while he was with Universität Tübingen, Germany, and some of the work was done while the first author was later with the University of Hertfordshire, Hatfield, UK.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2009.10.014

merely how the trees are drawn." Hence, our proposed algorithms can be seen as an attempt to automize finding "pleasing displays" of tanglegrams which biologists find useful and informative to compare phylogenies.

Recent related works. The conference version of this paper seems to have triggered a lot of algorithmic research on tree comparison. We briefly review some of those papers in the following to indicate the different research directions.

- M.S. Bansal *et al.* [1] define a generalized version of one of the problems we consider and show how their problem can be transformed to ours to solve it $\mathcal{FPT}$ time.
- Another generalization (to non-binary trees) is discussed in [35]. Furthermore, an approach by integer linear programming (ILP) is discussed.
- Further ILP-based papers have been written by several groups of researchers [4,8,30]. They also provide some experimental validations of their algorithms.
- Very recently, a different approach to the fixed-parameter tractablility of tanglegram problems, including giving experimental validation of their results, was described in the paper *A faster fixed-parameter approach to drawing binary tanglegrams* by S. Böcker, F. Hüffner, A. Truss and M. Wahlström, presented at IWPEC 2009.

Further relevant recent algorithmic work on tanglegram problems include [28] who provide an algorithm for the planar layout problem, called *drawability problem* later on in this paper, worse than ours in terms of running time (quadratic versus linear).

Definitions. In this paper, we only consider rooted unordered binary trees. By having defined a root, we have implicitly defined a direction of the edges, w.l.o.g., from the root down to the leaves. Hence, we can speak about the parent and the children of a node. The term "unordered" means that the order of the children of a node does not matter.

Let $L(T)$ denote the leaves of a tree T. A linear order $<$ on $L(T)$ is called *suitable* if T can be embedded into the plane such that $L(T)$ is mapped onto a straight line (*layer*) in the order given by $<$. A *two-tree* $(T_1, T_2; M)$ is given by a pair of rooted unordered binary trees (T_1, T_2) together with a perfect matching $M \subseteq L(T_1) \times L(T_2)$, where the matching is given by a bijective labeling $\lambda_i : L(T_i) \to \Lambda$ with $(\ell_1, \ell_2) \in M$ iff $\lambda_1(\ell_1) = \lambda_2(\ell_2)$.

A *drawing* of $(T_1, T_2; M)$ is given by two suitable linear orders $<_1$ and $<_2$ on $L(T_1)$ and $L(T_2)$, respectively. We assume that a drawing is *realized* by embedding $L(T_1)$ and $L(T_2)$ into two parallel lines L_1 and L_2, so that all nodes of T_i lie within one of the half-planes described by the line L_{3-i}, for $i = 1, 2$. Since the trees T_1 and T_2 can be embedded in the half-planes without crossings, only matching edges may cross.

Observation 1. The number of crossings is only dependent on the drawing, but independent of the chosen realization.

In other words, this quantity is a purely combinatorial object, irrespective of the concrete drawing. This is a situation that is similar to many other crossing problems. Due to our observation, we may define: Let $cr(T_1, T_2, M, <_1, <_2)$ denote the number of crossings in the drawing of $(T_1, T_2; M)$ given by $<_1$ and $<_2$. A two-tree $(T_1, T_2; M)$ is called *drawable* if $cr(T_1, T_2, M, <_1, <_2) = 0$ for some linear orders $<_1$ and $<_2$.

Simple examples. Let us clarify our definitions by some simple examples and easy results.

Lemma 1. *All two-trees with up to three leaves are drawable.*

Proof. The lemma is evident for two-trees with one leaf.

Consider a two-tree with two leaves. Even after fixing the order of leaves on one side, the leaves of the other side may be re-ordered if necessary to show up in the same order.

Finally, consider a two-tree with three leaves labeled a, b, c. There are four cases to consider:

- a and b have a common parent p_1 in T_1 and a common parent p_2 in T_2.
- a and b have a common parent p_1 in T_1 and b and c have a common parent p_2 in T_2.
- b and c have a common parent p_1 in T_1 and a and b have a common parent p_2 in T_2.
- b and c have a common parent p_1 in T_1 and a common parent p_2 in T_2.

In all cases, the order a, b, c can be realized without crossings, see Fig. 1.

Very simple non-drawable two-trees are shown in Fig. 2. Due to the previous lemma, these are the smallest examples of non-drawable two-trees.

Auxiliary notations. Since we like to talk about left and right subtrees and these notions usually depend on the parent's position, let us fix the following convention: We assume that all our trees are drawn either downwards (the *upper tree*) or

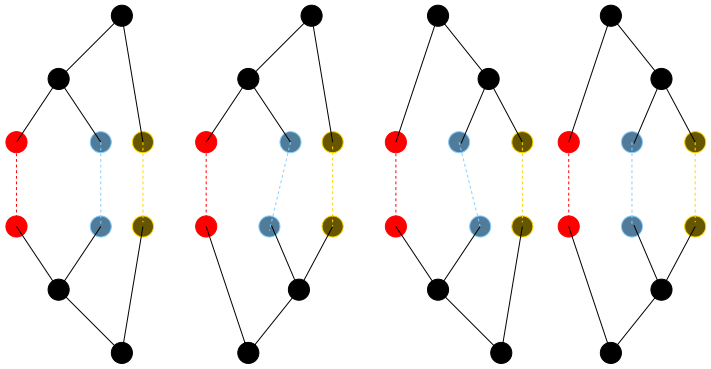

Fig. 1. Four situations for two-trees with three leaves.

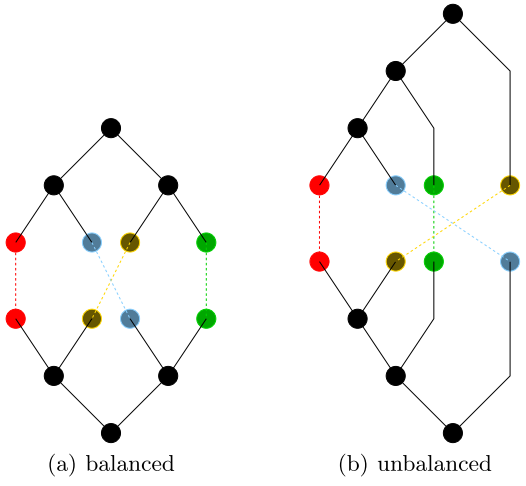

Fig. 2. The two types of contradicting quadruple two-trees.

upwards (the *bottom tree*); then, "left" and "right" refers to how an observer would name these relative directions when viewing such a drawing. More specifically, in a two-tree (T_1, T_2), T_1 is the upper and T_2 is the bottom tree. In a tree $T = (V, E)$ with root r, the notion of a *least common ancestor* $lca(X)$ of a non-empty set $X \subseteq V$ is well defined. More precisely, for any node x in T, the set of *ancestors* of x (in T) is the set of all nodes on the unique path between x and the root r (including x and r). Hence, if $X = \emptyset$, $lca(X) = \emptyset$, and if $X = \{x\}$, then $lca(X) = \{x\}$. For a pair of different nodes $\{x, y\}$, $lca(\{x, y\})$ is a one-element set containing p, the node that is, among all common ancestors of x and y (observe that the set of common ancestors of x and y is non-empty, since the root is always in this set), the one that is farthest off from the root. For $|X| > 3$, $lca(X) = \bigcup_{x,y \in X, x \neq y} lca(\{x, y\})$. Given $X \subseteq V$, let the *ancestral tree* $T\langle X\rangle = (V', E')$ be the given by $V' = X \cup lca(X)$ and $xy \in E'$ iff there is a path P from x to y in T and no other vertex $z \in V'$ is on P. Conversely, for $x \in V$, the *descendants tree* $T[x]$ is the graph induced by all vertices y of T such that the path from y to the root r contains x, i.e., x is an ancestor of y. So, y is a *descendant* of x iff x is an ancestor of y. Furthermore, if the embedding is fixed, we may talk about the *left* and the *right child* of an inner node x, written l_x and r_x, respectively. We can use such notations also when the drawings are only partially fixed, as it will be the case in many algorithms. If the embedding is not fixed, we will rather employ the notation $\chi(x)$ to denote the set of children of an inner node x.

Problem formulation. Consider a two-tree $(T_1, T_2; M)$. Let $cr(T_1, T_2, M, <_1, \cdot)$ denote the minimum value of $cr(T_1, T_2, M, <_1, <_2)$ for all suitable orders $<_2$, and $cr(T_1, T_2, M, \cdot, \cdot)$ denote the minimum value of $cr(T_1, T_2, M, <_1, <_2)$ for all suitable orders $<_1$ and $<_2$. We will discuss the following problems:

- An instance of TWO-TREE CROSSING MINIMIZATION (TTCM) is given by a two-tree $(T_1, T_2; M)$, and the parameter, a positive integer k.
  Is $cr(T_1, T_2, M, \cdot, \cdot) \leqslant k$?
- An instance of ONE-TREE CROSSING MINIMIZATION (OTCM) is given by a two-tree $(T_1, T_2; M)$, a suitable fixed order $<_1$ on $L(T_1)$, and the parameter, a positive integer k.
  Is $cr(T_1, T_2, M, <_1, \cdot) \leqslant k$?

- An instance of TWO-TREE DRAWING BY DELETING EDGES (TTDE) is given by a two-tree $(T_1, T_2; M)$, and the parameter, a positive integer k.
  Is there a set $M' = L'_1 \times L'_2 \subseteq M$ with $|M'| \leqslant k$ such that the two-tree $(T_1\langle L(T_1) \setminus L'_1\rangle, T_2\langle L(T_1) \setminus L'_2\rangle)$ is drawable?
- An instance of ONE-TREE DRAWING BY DELETING EDGES (OTDE) is given by a binary tree T_1 with leaf labels Λ, a linear ordering $\prec$ on Λ, and the parameter, a positive integer k.
  Is there a label set $L \subseteq \Lambda$ with $|L| \leqslant k$ such that the tree $T_1\langle \Lambda \setminus L\rangle$ can be drawn without crossings in the plane, so that the leaves in $\Lambda \setminus L$ are arranged according to the ordering $\prec$ on some line?

Observe that TWO-TREE DRAWING BY DELETING EDGES and ONE-TREE DRAWING BY DELETING EDGES are the natural edge deletion variants of TWO-TREE CROSSING MINIMIZATION and ONE-TREE CROSSING MINIMIZATION, respectively, keeping in mind that only matching edges between leaves might cross at all.

Application. A possible application from bioinformatics for the variant ONE-TREE CROSSING MINIMIZATION is that for a known species tree different gene trees should be compared to the species tree. The more general variant TWO-TREE CROSSING MINIMIZATION supports tasks like: (visually) compare different construction methods for phylogenetic trees for some data set (for example neighbor joining versus parsimony methods [23]) or compare multiple gene trees [24].

Parameterized complexity. While we assume that the reader is comfortable with basic notions of classical complexity theory like $\mathcal{NP}$-completeness, we will provide some notions necessary from the flourishing but yet less known area of parameterized complexity. More details can be found in existing textbooks of that field, most notably [11].

$\mathcal{NP}$-hard computational problems are ubiquitous in bioinformatics. One approach to overcoming this difficulty is to devise algorithms that can solve arbitrary instances of such a problem under the restriction that a certain entity, called the parameter, is small. This concept is usually formalized as follows: Problem instances are elements of $\Sigma^* \times \mathbb{N}$, and an instance $I = (w, k)$ is to be decided in time $\mathcal{O}(p(|w|)f(k))$, where p is a polynomial (whose degree does not depend on the parameter k) and f is an arbitrary function. Problems that can be solved within such a time restriction are called *fixed parameter tractable*, or in $\mathcal{FPT}$, for short.

Notice that a classical problem might possess various parameterizations. From a practical perspective, this approach to dealing with $\mathcal{NP}$-hard problems makes sense in particular if the chosen parameter is small in the circumstances of the application. In problems related to graph drawing, the number of crossings offers a good choice: if this number is too big, this kind of drawing would be discarded as a useful method to visually represent information. Several psychological experiments have shown that humans are not capable following lines with many crossings, in particular if the crossing angles become acute. The tanglegram problem studied in this paper offers another interpretation: if the parameter (again, the number of crossings in a tanglegram drawing) gets very big, this means that the two phylogenies under consideration completely disagree, which would point to a serious biological problem within at least one of the data sets.

Related problems. A related important problem from graph drawing [10] is the TWO-SIDED CROSSING MINIMIZATION problem (TSCM) for bipartite graphs, where the vertices within each layer are connected only to vertices of the other layer. The main differences are that the vertices might have more than one incident edge and that no trees restrict the possible orderings. TWO-SIDED CROSSING MINIMIZATION is $\mathcal{NP}$-complete, and the problem remains $\mathcal{NP}$-complete even if the order of one of the layers is fixed [18] (ONE-SIDED CROSSING MINIMIZATION(OSCM)). Both problems are fixed-parameter tractable, see [12,15,14] in chronological order. Only if both orders are fixed, then there are quite efficient polynomial-time algorithms for counting the number of crossings, see [3]. In the case of a binary tree with n leaves, there are exactly 2^{n-1} different leaf orders implied by different orderings of the subtrees. This is in contrast to the $n!$ permutations which are possible in OSCM.

Similarly related is the problem of finding an embedding of a graph in the plane that minimizes the number of crossings; this problem remains $\mathcal{NP}$-complete even if the degree of the graph is bounded by three. Notice that a two-tree (plus matching edges) obeys this degree bound. Moreover, in that case, the crossing minimization problem is known to be fixed-parameter tractable [19].

This is also true for the problems TWO-LAYER PLANARIZATION and ONE-LAYER PLANARIZATION [17], which are the natural edge deletion variants of TWO-SIDED CROSSING MINIMIZATION and ONE-SIDED CROSSING MINIMIZATION, respectively, see [13,20, 21,34] for results from a parameterized perspective.

Our main results.

- We improve on the dynamic programming approach exhibited in [16] to solve OTCM in time $\mathcal{O}(n \log n)$.
- We give a linear-time algorithm for deciding drawability of two-trees.
- We prove $\mathcal{NP}$-completeness for TTCM.
- We exhibit a combinatorial characterization of drawability based on the drawability of (all) sub-two-trees with four leaves. This can be seen as the mathematical main result of this paper, since in particular the results listed in the following item are then pretty straightforward.
- We show that the problems OTDE, TTDE and TTCM are fixed-parameter tractable.

This list of results also describes the overall organization of this paper.

2. Two cases that can be solved in polynomial time

2.1. The OTCM problem

Let $(T_1, T_2; M)$ be a two-tree with a fixed suitable order $<_1$ on $L(T_1)$. The task is to find a suitable order $<_2$ on $L(T_2)$ that minimizes the number of crossings of matched edges. We are going to give a dynamic programming solution to OTCM. Therefore, notice that any inner node v of T_2 defines a subproblem in the following sense: let L be the leaves from T_1 that are matched to leaves from $L(T_2[v])$; then, consider the two-tree

$$(T_1, T_2)[v] = \big(T_1\langle L\rangle, T_2[v], M \cap \big(L \times L\big(T_2[v]\big)\big)\big),$$

with the order $<_1$ restricted to L. Consider an inner node v of T_2. Let x, y be the two children of v in T_2. Then, $cr(T_2(x, y))$ denotes the number of pairwise crossings of the matching edges incident with leaves from $L(T_2[x])$ and $L(T_2[y])$, assuming that x is drawn left of y. Note that the total number of crossings for a certain fixed embedding where l_v and r_v are the left and right children of node v can be expressed as $\sum_v cr(T_2(l_v, r_v))$. Hence, we can express the minimum crossing number by the following recursion:

$$cr\big((T_1, T_2)[v], <_1, \cdot\big) = \sum_{x\in\chi(v)} cr\big((T_1, T_2)[x], <_1, \cdot\big) + \min\Big\{\big(cr\big(T_2(x, y)\big), cr\big(T_2(y, x)\big)\big) \;\Big|\; x, y \in \chi(v),\ x \neq y\Big\}$$

This recursion can be solved in a naive way in time $\mathcal{O}(n^2)$, as follows:

Firstly, we show how to compute $cr(T_2(x, y))$, with $x, y \in \chi(v)$, $x \neq y$. We denote $|L(T_2[x])| = n_x$ and $|L(T_2[y])| = n_y$ and assume, w.l.o.g., that $n_x \leqslant n_y$.

$cr(T_2(x, y))$ can be expressed as the sum over all $\ell \in L(T_2[x])$, where each term gives the number of $r \in L(T_2[y])$ to the left of ℓ; we call this number the *rank* of ℓ. This sum can be determined by a simple merge of the two lists of leaves in time linear in $|L(T_2[v])| = n_x + n_y$. The sum $cr(T_2(y, x))$ to be compared with $cr(T_2(x, y))$ can be computed by $cr(T_2(y, x)) = {n_v}^2 - cr(T_2(x, y))$.

Since the subtrees $T_2[x]$ and $T_2[y]$ are disjoint, the recursion for the runtime complexity reads $T(n) = T(n_x) + T(n_y) + c\cdot n$ for c constant and the number of leaves $n = n_x + n_y$ for root r with children x and y. Hence, the total time complexity is $\mathcal{O}(n^2)$.

Alternatively, we can implement the ordered sets $L(T_2[x])$ and $L(t_2[y])$ as level-linked (2–4) trees as being described in [29], Section III.5.3.3. We need to insert the elements of the smaller, say $L(T_2[x])$, into the larger one, say $L(T_2[y])$, compute the rank of each element of $L(T_2[x])$ and build the new larger level-linked (2–4) tree for $L(T_2[v])$. Theorem 14 in Section III.5.3.3 of [29] states that this can be done in time $\mathcal{O}(n_x \log((n_x + n_y)/n_x))$. The new recursion now reads $T(n) = T(n_x) + T(n_y) + cn_x \log((n_x + n_y)/n_x)$ where $n = n_x + n_y$ and $n_x \leqslant n_y$. By induction we can show that $T(n) = c\cdot n\log n$ for a suitable chosen constant c. The base is done with $T(1) \leqslant c \leqslant \mathcal{O}(1)$. The induction step follows from:

$$\begin{aligned} T(n) &= T(n_x) + T(n_y) + cn_x \log\big((n_x + n_y)/n_x\big) \\ &= cn_x \log n_x + cn_y \log n_y + cn_x \cdot \big(\log(n_x + n_y) - \log n_x\big) \\ &\leqslant cn_y \log n + cn_x \log n \\ &\leqslant cn \log n. \end{aligned}$$

Theorem 1. *In time $\mathcal{O}(n \log n)$, we can solve the* OTCM *problem, where n is the number of leaves.*

2.2. An efficient algorithm for the non-crossing case

Of course, we can use the previous theorem to produce a drawing of a two-tree in the special case that no crossing edges are necessary. However, we can use a specific machinery from graph drawing to obtain an improved algorithm for this special case.

Theorem 2. *Given a two-tree $(T_1, T_2; M)$, its drawability can be decided in linear time. If the two-tree is drawable, then a drawing can be produced in the same amount of time.*

Proof. The two input trees together with the matching edges can be naturally directed upward having the two roots as single source and sink respectively. Then, we can directly apply the linear time algorithm for upward planarity of acyclic digraphs with a single source [5].

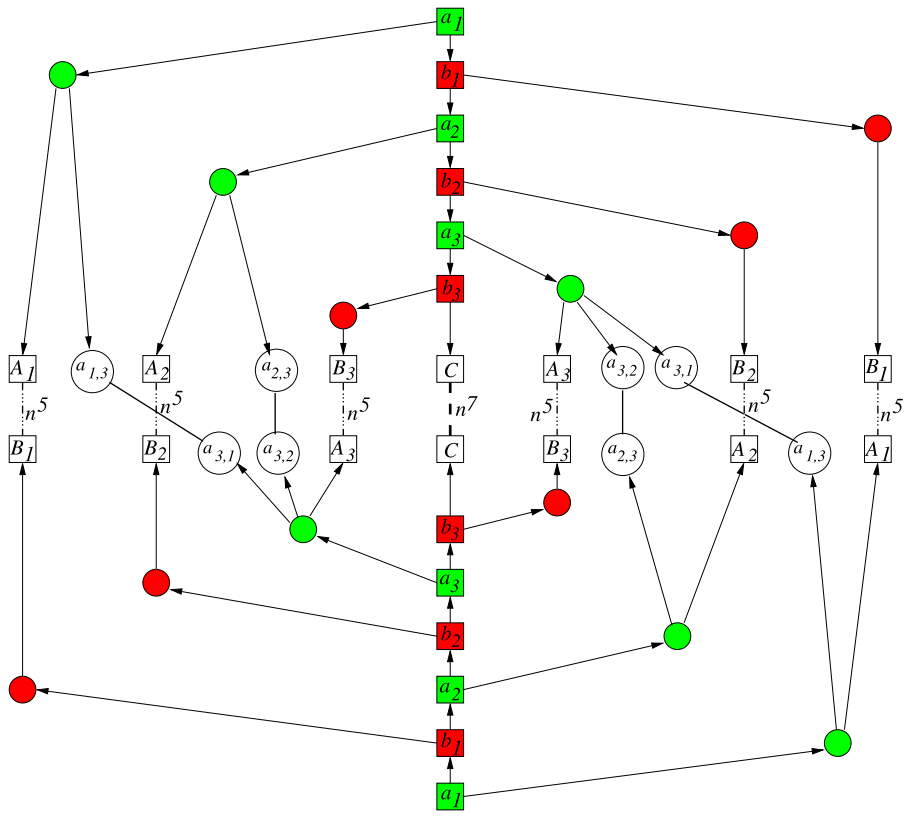

Fig. 3. The construction of the two trees representing a small MAXCUT instance with just 3 vertices and 2 edges, where vertex 3 is connected to vertices 1 and 2. Observe that a'_v and b'_v are represented by green and red empty circle nodes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. The general case for TTCM is hard

Theorem 3. TWO-TREE CROSSING MINIMIZATION *is $\mathcal{NP}$-complete.*

Proof. Membership in $\mathcal{NP}$ is clear. Next, we give a reduction of the MAXCUT problem with unit weights. The MAXCUT problem is to partition the vertex set V into V_1 and V_2 for a given graph $G = (V, E)$, such that $|\{e = (v, w) \in E$ with $v \in V_1$ and $w \in V_2\}|$ is maximized. For a pictorial explanation of this reduction, we refer to Fig. 3.

So, let $G = (V, E)$ with $V = \{1, \ldots, n\}$ be an instance of MAXCUT. From G, we construct an instance of the TWO-TREE CROSSING MINIMIZATION problem, so that we have a "backbone"-path $a_1, b_1, \ldots, a_n, b_n, C$ in both of the trees T_1, T_2, which, with the leaf-layers, partitions the drawing area into four parts that later give us the membership of each vertex to V_1 or V_2, respectively. Let a_1 be the root in each tree, C is one of the leaves. For each vertex $v \in V$, we connect two representative nodes a'_v, b'_v to the corresponding backbone nodes a_v, b_v in each tree. Moreover, to each of those representative nodes a'_v, b'_v, we have to connect further representatives in the leaf-layer, say A_v, B_v; A_v-leaves shall be matched to the B_v-leaves of the other tree and vice versa with n^5 edges for each $v = 1, \ldots, n$. For each edge $e = \{v, w\} \in E$, we create leaf-vertices $a_{v,w}$ and $a_{w,v}$ and connect both pairwise by matching edges. Furthermore, we connect a'_v with the leaf nodes $a_{v,w}$ for all $e = \{v, w\} \in E$. We also connect the two C leaves of the backbone to each other via n^7 edges. To make the trees binary, every vertex that is connected to more than one other vertex is substituted by a small binary subtree with an appropriate number of leaves. We can observe the following facts:

1. In any optimum solution, there is no crossing between edges adjacent to C-nodes and $(A_v - B_v)$-edges.
2. In any optimum solution, the a'_v- and b'_v- vertices are on different sides of the backbone in both trees. The side for a'_v in T_1 is different from that in T_2.
3. In any optimum solution, the number of edges $(a_{v,w}, a_{w,v})$ crossing the backbone connection is minimized, and the number of edges $(a_{v,w}, a_{w,v})$ which do not cross the backbone is maximized.

Now, we define $V_1 = \{i \mid a_v$ is on the left-hand side of the backbone in $T_1\}$ and $V_2 = \{i \mid a_v$ is on the right-hand side of the backbone in $T_1\}$ for splitting V into two disjoint sets V_1 and V_2. From our observations, we can see that this is an

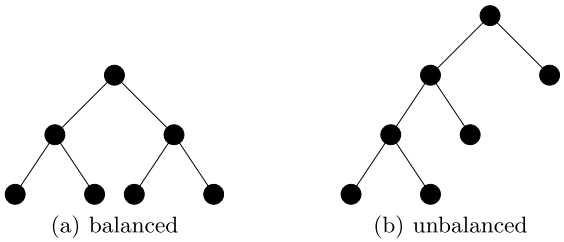

Fig. 4. The two types of quadruple trees to be analyzed.

optimal solution for the MAXCUT problem. Obviously, this construction can be built in polynomial time. Hence the reduction of MAXCUT to TTCM is completed.

Remark 1. The astute reader might have observed that the parameter k of TTCM gets really big in the reduction of the previous theorem. This is in fact typical for $\mathcal{NP}$-hard problems which are then classified into $\mathcal{FPT}$. The parameter with respect to which $\mathcal{FPT}$-membership is shown must reflect parts of the combinatorial hardness of the problem we start with (here, MAXCUT); taken to the extreme: if k would become constant under the reduction, then the $\mathcal{FPT}$-algorithm for TTCM we will present later would deliver a (deterministic) polynomial-time algorithm for MAXCUT, which in turn would imply that $\mathcal{NP}$ equals $\mathcal{P}$.

4. Quadruple trees

To establish our non-trivial algorithmic results, we need a complete analysis of what happens if we restrict ourselves to the case of two trees T_1 and T_2 each having four leaves labeled a, b, c, d. We will also refer to such small trees as *quadruple trees* and the four leaf labels are then called a *quadruple*. Since our trees $T = T_i$ are binary, we can have only two different cases: T has depth two and is hence *balanced*; T has depth three and is *unbalanced*, see Fig 4. We will show that there are only two types of non-drawable quadruple trees as depicted in Fig. 2.

4.1. Analyzing quadruple trees

Let us first analyze the balanced trees. If we assume a labeling a, b, c, d in this sequence along the leaves of a balanced tree (in the sequence as shown in Fig. 4), then let $[abcd]$ denote the different labelings of leaves that can be obtained by different drawings of that particular tree, i.e., by redefining (swapping) the left/right child relations in that tree. We can observe the following possibilities:

$$B_1 = [abcd] = \{abcd, abdc, bacd, badc, cdab, cdba, dcab, dcba\},$$
$$B_2 = [acbd] = \{acbd, acdb, bdac, bdca, cabd, cadb, dbac, dbca\}, \quad \text{and}$$
$$B_3 = [adbc] = \{adbc, adcb, bcad, bcda, cbad, cbda, dabc, dacb\}.$$

The classes are chosen in a way that *acbd* is the lexicographically first label not contained in B_1, and *adbc* is the first label not in B_1 nor B_2.

So, starting with a balanced quadruple tree T with root r and inner nodes x, y (where first x is drawn left of y), where the children i, j of x and k, l of y are labeled a, b, c, d (in that order), we can see that also $abdc \in [abcd]$ by swapping k and l in the described realization T of the leaf sequence *abcd*. Similarly, by swapping i and j, the leaf sequence *bacd* can be obtained from T. If we swap both i and j, and k and l, we can see that $badc \in [abcd]$. If we swap x and y, it can be seen that $cdab \in [abcd]$. The other three leaf sequences *dcba*, *cdba*, or *dcab*, respectively, can be also obtained by swapping x and y in the described realizations of *abdc*, *bacd*, or *badc*, respectively.

Observe that $B_1 \cap B_2 = B_1 \cap B_3 = B_2 \cap B_3 = \emptyset$.

Lemma 2. *B_1, B_2, B_3 are the three mutually disjoint equivalence classes of leaf labelings for balanced trees that together exhaust all* 24 *permutations of a, b, c, d.*

If we assume that both T_1 and T_2 are balanced quadruple trees, then we can conclude from Lemma 2:

Proposition 1. *Let T_1 and T_2 be balanced quadruple trees. Let L_i^1 (L_i^2, resp.) be the two two-element label sets that label leaves reachable from the first* (*second, resp.*) *child of the root of T_i. Then, the pair (T_1, T_2) can be drawn without crossings iff $L_1^1 = L_2^1$ or $L_1^1 = L_2^2$. Hence, non-drawability in the balanced quadruple tree case means that, with $L_1^1 = \{x, y\}$, $x \in L_2^1$ and $y \in L_2^2$.*

Let us now turn towards unbalanced quadruple trees. We first of all list all possible $[wxyz]$ classes. Note that the only symmetry is here within the labeling of the first two leaves (as drawn in Fig. 4). Therefore, we get $24/2 = 12$ possible classes of labels:

$$U_1 = [abcd] = \{abcd, bacd, cabd, cbad, dabc, dbac, dcab, dcba\},$$
$$U_2 = [abdc] = \{abdc, badc, cabd, cbad, cdab, cdba, dabc, dbac\},$$
$$U_3 = [acbd] = \{acbd, bacd, bcad, cabd, dacb, dbac, dbca, dcab\},$$
$$U_4 = [acdb] = \{acdb, bacd, bcad, bdac, bdca, cadb, dacb, dcab\},$$
$$U_5 = [adbc] = \{adbc, badc, bdac, cadb, cbad, cbda, cdab, dabc\},$$
$$U_6 = [adcb] = \{adcb, badc, bcad, bcda, bdac, cadb, cdab, dacb\},$$
$$U_7 = [bcad] = \{abcd, acbd, bcad, cbad, dabc, dacb, dbca, dcba\},$$
$$U_8 = [bcda] = \{abcd, acbd, adbc, adcb, bcda, cbda, dbca, dcba\},$$
$$U_9 = [bdac] = \{abdc, adbc, bdac, cabd, cadb, cbda, cdba, dbac\},$$
$$U_{10} = [bdca] = \{abdc, acbd, acdb, adbc, bdca, cbda, cdba, dbca\},$$
$$U_{11} = [cdab] = \{acdb, adcb, bacd, badc, bcda, bdca, cdab, dcab\}, \quad \text{and}$$
$$U_{12} = [cdba] = \{abcd, abdc, acdb, adcb, bcda, bdca, cdba, dcba\}.$$

By inspection, one can see:

Lemma 3. *U_1, U_6, U_{10} are three mutually disjoint equivalence classes of leaf labelings for unbalanced trees that together exhaust all 24 possible permutations of a, b, c, d. (There are three more groups of three U-sets with this property.)*

Proposition 2. *Let T_1 and T_2 be unbalanced quadruple trees. Assume (w.l.o.g.) that T_1 and T_2 are drawn as in Fig. 4 (of course, T_2 must be flipped to get the root at the bottom), with labels a, b, c, d attached to the leaves of T_1 in that order. Then, the pair (T_1, T_2) is drawable iff either the third leaf of T_2 is not labeled c or the last leaf of T_2 is not labeled with one of the first two labels of T_1.*

Hence, such a pair cannot be drawn iff the third leaf coincides and the last leaf of T_2 is among the first two leaves of T_1. Comparing the classes of unbalanced trees with those of balanced trees, one can see that for all i, j, $B_i \cap U_j \neq \emptyset$, so that such a pairing can always been drawn.

Proposition 3. *Let T_1 and T_2 be quadruple trees, where T_1 is balanced and T_2 is unbalanced (or vice versa). Then, the pair (T_1, T_2) is drawable.*

Summarizing, we find that there are only two essentially different situations when a quadruple tree is non-drawable. These situations are depicted in Fig. 2.

4.2. Quadruples are all about inconsistencies

The following theorem shows that the drawability of a two-tree only depends on the drawability of the induced quadruple trees.

Theorem 4. *Let $(T_1, T_2; M)$ be a two-tree with labelings $\lambda_i : L(T_i) \to \Lambda$. Then, $cr(T_1, T_2, M, \cdot, \cdot) > 0$ if and only if there exists a quadruple $Q = \{a, b, c, d\} \subseteq \Lambda$ such that $cr(T_1\langle\lambda_1^{-1}(Q)\rangle, T_2\langle\lambda_2^{-1}(Q)\rangle, M \cap (\lambda_1^{-1}(Q) \times \lambda_2^{-1}(Q)), \cdot, \cdot) > 0$.*

Notice that $T_1\langle\lambda_1^{-1}(Q)\rangle$ describes the binary tree induced by the leaves Q (or, more precisely, $\lambda_1^{-1}(Q)$) within T_1. Besides Q, this binary tree contains as nodes all least common ancestors of sets of nodes from Q. Similar observations apply to $T_2\langle\lambda_2^{-1}(Q)>\rangle$. So, $(T_1\langle\lambda_1^{-1}(Q)\rangle, T_2\langle\lambda_2^{-1}(Q)>\rangle; M \cap (\lambda_1^{-1}(Q) \times \lambda_2^{-1}(Q)))$ describes a quadruple tree.

Within the proof of Theorem 4, we will present a recursive algorithm which either finds such a quadruple, or it provides a drawing of the two-tree. This algorithm will not only prove this structural result that gives a characterization of drawable two-trees in terms of forbidden substructures, but also provides the backbone of $\mathcal{FPT}$ algorithms that are presented in the following section.

Proof of Theorem 4. The "only-if"-part is obvious. We are going to show the "if-part" in what follows.

Let $(T_1, T_2; M)$ be the two-tree and Λ the set of labels associated to the leaves. Without loss of generality, $|\Lambda| > 1$.

We assume that each inner node provides links to its two children and in addition permanent and temporary information attached to each such link. The permanent information $p(\ell)$ attached to link ℓ is either L, R or $*$, meaning:

L	This link leads to the left child
R	This link leads to the right child
$*$	It is not yet determined if this link leads to the left or to the right child

The permanent information indicates a commitment of how the links are to be drawn (either forced or without loss of generality) in order to obtain a crossing-free embedding of the two-tree; once fixed, it will not be changed in later stages of the algorithm. The temporary information is only used to either produce further evidence that allows to make further commitments of the permanent information or to provide a contradictory quadruple.

In the initialization phase of our algorithm, we (arbitrarily) set $p(\ell_1) = L$ and $p(\ell_2) = R$ for the two links emanating from the root of T_2. All other permanent information is set to $*$. This defines the function p (that we use both for T_1 and for T_2) in the very beginning. Thus initiated, we call the procedure embed(T_1, T_2, p). The way how the permanent information is initiated and updated shows that the following is always true:

Claim 1. *Let $\mathfrak{n}$ be an inner node with emanating links ℓ_1 and ℓ_2. Then, $p(\ell_1) = L$ iff $p(\ell_2) = R$, and $p(\ell_1) = R$ iff $p(\ell_2) = L$.*

The temporary information $t(\ell)$ attached to link ℓ is either l, r or m, meaning:

l	All links ℓ' below (i.e., in direction to the leaves) satisfy $t(\ell') = l$
r	All links ℓ' below satisfy $t(\ell') = r$
m	Mixed case: some links below are marked l and some are marked r

In actual fact, in the initialization phase, we will assign also $*$ to some links, i.e., $t(\ell) = *$ could happen, this way explicitly denoting a yet undecided value. Moreover, we might assign $t(\ell) = e$ signaling an error case.

The temporary information is processed in a bottom-up fashion from the leaves to the root as follows:

1. As described in Algorithm 1 in detail, the links leading to the leaves of the tree to be processed are assigned either l or r (or the undefined value $*$).
2. Let $\mathfrak{n}$ be an inner node (besides the root) where to both links ℓ_1 and ℓ_2 emanating to its two children, the temporary information has been assigned. Then, to the link ℓ that leads to $\mathfrak{n}$ we assign $t(\ell)$ according to the following table:

$t(\ell_1)$	l	l	l	r	r	r	m	m	m
$t(\ell_2)$	l	r	m	l	r	m	l	r	m
$t(\ell)$	l	m	m	m	r	m	m	m	e

Here, e signals an error case: we have found a quadruple situation corresponding to the balanced tree case in Fig. 2; this is detailed in the next paragraph. Hence, there is no way of finding a crossing-free embedding of the two-trees, and we can abort here. So, we need not describe how e is further propagated.

The error situation encountered in item two above is obtained by the following scenario: There exists four leaf labels $Q = \{a, b, c, d\}$, such that a, b are found in the left subtree of T_2 (seen from the root), i.e., $a, b \in L(T_2[y_L])$ in the notation of Algorithm 1, and $c, d \in L(T_2[y_R])$. In T_1, Q induces the following situation: there are three inner nodes $\mathfrak{n}_1$, $\mathfrak{n}_2$ and $\mathfrak{n}$, such that $\{\mathfrak{n}_1\} = lca(\{a, c\})$, $\{\mathfrak{n}_2\} = lca(\{b, d\})$, and $\{\mathfrak{n}\} = lca(\{\mathfrak{n}_1, \mathfrak{n}_2\})$. Consider the temporary information sequence $t_1, t_2, \ldots, t_q$ corresponding the link sequence from a to $\mathfrak{n}_1$ in T_1. We observe that there must be a $1 \leqslant J \leqslant q$ such that for all $1 \leqslant j \leqslant J$, $t_j = \ell$, and if $J < q$, we also find that for all $J < j \leqslant q$, $t_j = m$. Similarly, the temporary information sequence corresponding to the link sequence from c to $\mathfrak{n}_1$ in T_1 starts with r. Therefore, the temporary information sequence corresponding to the link sequence from $\mathfrak{n}_1$ to $\mathfrak{n}$ in T_1 only contains m. A symmetric argument applies to the temporary information sequences involving $\mathfrak{n}_2$ instead of $\mathfrak{n}_1$. In particular, both links leading from $\mathfrak{n}$ to the children carry the temporary information m. Hence, the error case is observed. Moreover, the contradicting quadruple tree is explicitly displayed. So, both in the lower and in the upper tree, the subtree that is induced by the labels a, b, c, d is a balanced quadruple tree, and the shape of both quadruple trees is contradictory as explained in Fig. 2.

Interestingly, we can also update the permanent information of two siblings. Let $\mathfrak{n}$ be an inner node where to both links ℓ_1 and ℓ_2 emanating to its two children, the temporary information has been assigned. We update $p(\ell_1)$ and $p(\ell_2)$ according to the following table:

$t(\ell_1)$	l	l	l	l	l	l	l	l	l
$t(\ell_2)$	l	l	l	r	r	r	m	m	m
$p(\ell_1)$	L	R	$*$	L	R	$*$	L	R	$*$
$p(\ell_2)$	R	L	$*$	R	L	$*$	R	L	$*$
$p(\ell_1)$	L	R	$*$	L	E	L	L	E	L
$p(\ell_2)$	R	L	$*$	R	E	R	R	E	R

Algorithm 1 Procedure "embed-TT".

Require: A two-tree $(T_1, T_2; M)$ and a permanent link information function p.

Ensure: YES iff a crossing-free drawing of $(T_1, T_2; M)$ respecting p can be obtained. Moreover, either (implicitly) a crossing-free drawing of $(T_1, T_2; M)$ respecting p is found or a contradictory quadruple two-tree is produced.

Let x be the root of T_2.
if $(T_1, T_2; M)$ has at most four leaves **then**
  return answer by table look-up (produced according to Section 4.1)
else if x has only one child **then**
  Delete x to produce T_2'.
  Modify p and M accordingly, yielding p' and M'.
  return embed-TT(T_1, T_2', M', p');
else
  {Let ℓ_1 and ℓ_2 be the two links emanating from the root x of T_2.}
  if $p(\ell_1) = *$ **then**
    $p(\ell_1) := L$ and $p(\ell_2) := R$ {without loss of generality}.
  end if
  {Let $\ell_L = xy_L$ with $p(\ell_L) = L$ and $\ell_R = xy_R$ with $p(\ell_R) = R$.}
  Let $L_L := L(T_2[y_L])$ and $L_R := L(T_2[y_R])$.
  Let $L_l := \{u \in L(T_1) \mid \exists u' \in L_L : (u, u') \in M\}$ and $L_r := L(T_1) \setminus L_l$.
  {Initialize the bottom-up computation of new temporary information:}
  for all links $\ell = uv$ of T_1 where v is closer to the root than u **do**
    if $u \in L(T_1)$ **then**
      $t(\ell) := z \in \{l, r\}$ such that $u \in L_z$.
    else
      $t(\ell) := *$ {for links between inner nodes}
    end if
  end for
  Update the temporary and permanent information within T_1 as in the proof.
  if contradiction is reached **then**
    Report contradictory quadruple two-tree (see the proof of Theorem 4).
    return NO
  else
    Let p_L be the permanent information p updated to cover the two-tree $(T_2[y_L], T_1\langle L_l\rangle; M_L)$ with $M_L^{-1} = M \cap (L_l \times L_L)$; similarly, define p_R, M_R.
    return embed-TT$(T_2[y_L], T_1\langle L_l >\rangle, M_L, p_L)$ $\wedge$embed-TT$(T_2[y_R], T_1\langle L_r\rangle, M_R, p_R)$
  end if
end if

Observe that there are more cases for assigning temporary information, with the roles of ℓ_1 and ℓ_2 being interchanged. Furthermore, notice that then the list of cases of assignments to ℓ_1 and ℓ_2 is complete with respect to the permanent information because of Claim 1. The table should be read as follows: the first four lines give the current values of t and p on the two links. The last two lines give the updated values of p. Here, an E signals that we found a contradictory situation; more specifically (as we will see below), we have found a quadruple situation corresponding to the unbalanced tree case in Fig. 2 Hence, there is no way of finding a crossing-free embedding of the two-trees, and we can abort here.

Observe that, while the temporary information is propagated bottom-up, there are only some (isolated) cases when previously unsettled permanent information is settled. Specifically, this is the case when one sibling is assigned temporary information ℓ (or r, resp.) and the other sibling is not assigned ℓ (or r, resp.). However, as the next claim shows, the "settled regions" will be finally connected. There are other cases when previously settled permanent information is *confirmed*, as for example in the very first column of the given table for t: no contradiction to the temporary information is found.

Claim 2. *Observe that the graph that is induced by the edges (links) to which non-$*$ permanent information has been attached to is a tree before and after each complete bottom-up tree processing (as described above). Moreover, if this induced tree is non-empty, then it also contains the root.*

Proof. The claim is obviously true at the very beginning. Let n be a first node where (in the bottom-up processing) permanent information is set to its outgoing links. This means that to the link ℓ_0 of its ancestor, m will be ascribed as temporary information. In order to avoid an error case, either r or ℓ must be assigned to the sibling of ℓ_0 as temporary information. If there has been previously assigned non-$*$ permanent information to ℓ_0, the claim follows. Otherwise, the table we set up for updating the permanent information shows that we will assign some non-$*$ permanent information at this stage to ℓ_0. This argument continues upwards in direction to the root, until either an error case was found (in that case, the bottom-up tree processing sees an early-abort and is therefore not considered in the formulation of the claim) or the region to which permanent non-$*$ information is attached to forms a tree (by applying the induction hypothesis). □

How to actually use the bottom-up processing of the temporary and permanent information is explained in Algorithm 1. Observe that the roles of the two trees alternate. We will make use of the following property of our algorithm:

Claim 3. *Each time that it is again say the upper tree's turn to get new temporary labels, the former root of that tree (and possibly more nodes) will no longer be taken into consideration.*

We still have to show that the two aborts (error cases) described above are indeed based on finding a contradictory quadruple two-tree as explained in Fig. 2. The proofs of the following two claims exhibit the details of our construction.

Claim 4. *Whenever an error occurs within the temporary label information update, we can exhibit a balanced quadruple two-tree.*

Proof. The reasoning showing this claim can be found right after our introduction of the temporary information update. □

Claim 5. *Whenever a contradiction is found between the temporary label information and the already existent permanent label information, we can exhibit an unbalanced quadruple two-tree. A conflict occurs here if there are two sibling links ℓ_1 and ℓ_2 emanating from an inner node such that the bottom-up algorithm infers from the temporary information an order of the links that is different from what was already stored in the permanent information.*

Proof. Without loss of generality, let us assume that ℓ_1 and ℓ_2 belong to the upper tree and that $p(\ell_1) = L$ and $p(\ell_2) = R$ and that $t(\ell_1) = r$ and $t(\ell_2) = \ell$. Let v be the node where both ℓ_1 and ℓ_2 emanate. Assume we observed a conflict say at the t_1th level of the recursion. There must have been a point of time (i.e., level) in the recursion that for the first time fixed $p(\ell_1) = L$ and $p(\ell_2) = R$. This point could not be the initialization due to Claim 3. Since we did not observe a conflict at level $t_0 = t_1 - 2$ of the recursion, we either fixed at time t_0 for the first time the permanent information that is at t_1 leading to a conflict or it got at least confirmed when processing level t_0. This fixing (or confirming) was obtained by using the temporary link information (at time t_0). So, at time t_0, there was a leaf labeled a reachable from ℓ_1 that then was also a label of a leaf in the left subtree of the lower tree. Let us now furthermore assume (again without loss of generality, by completely symmetric arguments in the other case) that the recursion branch where our observed conflict occurs is along recursing on the right subtree T_R^ℓ of the lower tree and on the left subtree T_L^u of the upper tree (where these subtrees were considered at level $t = t_0 + 1 = t_1 - 1$ of the recursion).

Consider some leaf labeled d in the right subtree T_R^u of the upper tree at level t. Naturally, d is found as a leaf label in the right subtree T_R^ℓ of the lower tree (referring to level t_0), since otherwise we would have observed non-drawability at an earlier stage. Let y be the root of T_R^ℓ. The position of d also permanently fixes the order of children links of y. Now, if the labels of the leaf descendants of the right child of y are only belonging to the label set of the leaves in T_R^u, then we will not get a contradiction at level t_1 as presumed. Hence, upon further recursing on T_L^u in step t_1, we will find both left and right descendants of y that carry labels that can be also found in T_L^u.

Let us further discuss the node v in the upper tree in what follows. There must be a descendant z of v (along ℓ_1) where one of the branches starting at z leads to a (and in fact we may assume all other leaves that can be reached from that branch carry labels that are in T_L^ℓ) and the other branch leads to some leaf with label c that can be also found as leaf label in the right tree at level t_1 of the recursion (since $t(\ell_1) = R$ at level t_1). Moreover, that descendant z is no longer "visible" at level t_1, since the "left descendants" of z do not find their counterparts in T_R^ℓ. Hence, $v \neq z$.

Moreover, by assumption on the temporary information of ℓ_2, there must be a leaf node labeled b that is reachable from ℓ_2; a leaf labeled b can be also reached in the lower tree from the left child of y.

This proves that the labels a, b, c, d as constructed above present a contradictory quadruple two-tree. □

5. Fixed-parameter tractability

Parameterized complexity and algorithmics is now an established way of dealing with hard problems that have a natural parameter in its definition. The idea is that, for small parameter values, we can get away with a polynomial-time algorithm, where the degree of the polynomial is independent of the parameter. This is the approach we take in this section.

Recall that a *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a fixed alphabet and $\mathbb{N}$ is the set of all non-negative integers. Therefore, each instance of the parameterized problem P is a pair (I, k), where the second component k is called the *parameter*. The language $L(P)$ is the set of all YES-instances of P. The parameterized problem P is said to be *fixed-parameter tractable* [11] if there is an algorithm that decides whether an input (I, k) is a member of $L(P)$ in time $f(k)|I|^c$, where c is a fixed constant and $f(k)$ is a recursive function independent of the overall input length $|I|$. The class of all fixed-parameter tractable problems is denoted by $\mathcal{FPT}$. Ignoring the polynomial part of a parameterized algorithm, we write $\mathcal{O}^*(f(k))$ to indicate the non-polynomial running time estimate.

5.1. Edge deletion variants

Theorem 4 shows an immediate result for the problem TWO-TREE DRAWING BY DELETING EDGES that is closely related to TTCM: Namely, we can translate any TWO-TREE DRAWING BY DELETING EDGES instance into 4-HITTING SET: simply cycle through all $\mathcal{O}(n^4)$ possible quadruple two-trees (given a concrete two-tree (T_1, T_2) with n leaves): If a quadruple two-tree is contradictory, then it corresponds to a hyperedge with four vertices, the leaf labels forming that quadruple. All n leaf

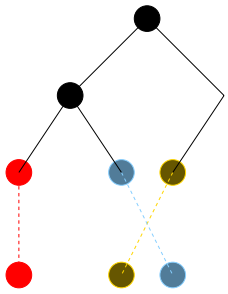

Fig. 5. The contradicting triple tree.

Algorithm 2 Try to embed a tree, called "embed-OT".

Require: A tree T_1 with ordered leaf set $(\Lambda, \prec)$ and a permanent link information function p.
Ensure: YES iff a crossing-free drawing of T_1 respecting p and $\prec$ can be obtained. Moreover, either (implicitly) a crossing-free drawing of T_1 respecting p and $\prec$ is found or a contradictory triple tree is produced.

if T_1 is empty **then**
  return YES
end if
Let r be the root of T_1.
Let $\ell \in \Lambda$ be the smallest label according to $\prec$.
Update p along the path P from the leaf labeled ℓ to the root.
{Setting $p(\ell_1) = L$ means setting $p(\ell_2) = R$ for the sibling ℓ_2 of ℓ_1 and vice versa.}
if contradicting permanent link information is found **then**
  return NO
else
  Delete ℓ from $(\Lambda, \prec)$ and from T_1.
  {Thereby, also the parent of ℓ in T_1 disappears by edge contraction; there will be no permanent information associated to the contracted edge.}
  return embed-OT$(T_1, \prec, p)$.
end if

labels together are the vertices of the hypergraph. Using known parameterized algorithms for 4-HITTING SET, see [20], we can thus show:

Corollary 1. TTDE *is solvable in* $\mathcal{O}^*(3.115^k)$ *time.*

Let us mention that we can similarly translate the variant ONE-TREE DRAWING BY DELETING EDGES of TTDE where one tree is fixed into 3-HITTING SET.

Some straightforward case analysis shows:

Lemma 4. *When the order of the leaf labels is fixed, then there is only one possible contradictory situation in a tree with three leaves, and this situation is shown in Fig.* 5.

Algorithm 2 shows:

Theorem 5. *In each* ONE-TREE DRAWING BY DELETING EDGES *instance* $(T, \prec)$ *that is not embeddable without crossings, there exists a triple* $\{a, b, c\}$ *of leaf labels which is, with its induced tree structure* $T(\{a, b, c\})$ *also not embeddable without crossings (when respecting* $\prec$*).*

With Theorem 5 at hand, we can translate any ONE-TREE DRAWING BY DELETING EDGES instance into 3-HITTING SET as follows: simply cycle through all $\mathcal{O}(n^3)$ possible quadruple two-trees (given a concrete tree T with n leaves and an ordering $\prec$ of the leaves): If a triple tree is contradictory, then it corresponds to a hyperedge with three vertices, the leaf labels forming that triple. All n leaf labels together are the vertices of the hypergraph.

With the help of known parameterized algorithms for 3-HITTING SET, see [36], we can thus show:

Corollary 2. ONE-TREE DRAWING BY DELETING EDGES *can be solved in time* $\mathcal{O}^*(2.08^k)$.

Notice that the efficient dynamic-programming algorithm derived for the related problem OTCM in Section 2 cannot be transferred to this problem. However, we have no proof for $\mathcal{NP}$-hardness for OTDE nor TTDE, either.

Observe that all two problems would benefit from further advances of parameterized algorithms for 3-HS and 4-HS, respectively.

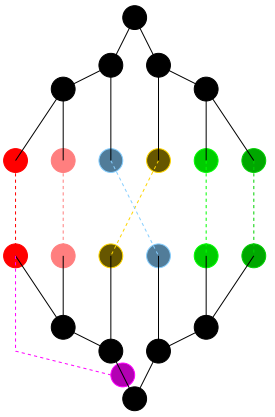

Fig. 6. Four contradicting quadruples, but only one crossing.

5.2. TWO-TREE CROSSING MINIMIZATION *is in* $\mathcal{FPT}$

This result is heavily based on the structural results of the previous section. More precisely, we will sketch a parameterized algorithm that branches on small contradicting structures (primarily, at contradicting quadruples) as long as these incur new crossings. In a second phase, we attempt at drawing the remaining two-tree with using a variant of the algorithm embed-TT, possibly finding new small contradicting structures. The validity of this approach relies on the fact that we are able to separate contradicting structures from the rest of the two-tree by attachment links that are described as follows.

This is not completely trivial, as the next example shows:

Example 1. Have a look at the two-tree depicted in Fig. 6. It contains four contradicting quadruples: take the two (brown and blue) middle leaves plus one of the reddish and one of the greenish leaves. However, once having branched at one of these quadruples and hence fixing the sequence of the two middle leaves in each tree will yield a situation where further branches (according to other contradicting quadruples) will not introduce any further crossings in the search tree branches that actually put the brown and blue leaves in-between the two other leaves of the quadruple. This means that we cannot reduce the parameter budget in each subsequent branch, which renders it impossible to immediately classify TTCM into $\mathcal{FPT}$.

Of course, this example can be enhanced to produce arbitrarily bad ratios between the number of bad quadruples we want to branch on and the number of crossings in at least one particular ordering (which will stay constant one).

In the shown example, transitivity comes at help: fixing one of the reddish leaves to be to the left of the two middle leaves also fixes the other one in this relative order due to the tree structures. Hence, looking for the transitive closure of a temporarily constructed ordering of the leaves is sometimes helpful.

We can however find a similar example where transitivity is not helpful: watch the pink dashed link at the bottom left of Fig. 6 that indicates a different tree structure (that could be analogously replicated in the other three "corners" of the graph); now, there will be no further commitments due to transitivity (and due to the tree structure) if we start with the "innermost" quadruple. So, we cannot simply ignore the indicated branching problems.

Let u be a node in a tree T. The *parent link* $\mathfrak{p}(u)$ is the unique edge leading to u in T; if u is the root, then there is no parent link to u. Let $V' \subseteq V(T)$. The set of all parent links of V' is $\mathfrak{P}(V') = \{\mathfrak{p}(u) \mid u \in V'\}$. The set of *attachment links* of a subgraph G of T is $\mathfrak{A}(G) = \mathfrak{P}(lca(V(G)) \cup V(G))$.

As above, we work with the permanent information $p(\ell)$ attached to an edge ℓ, initialized with $p = *$ and later gradually updated to either L or R. Sometimes, it is more convenient to think of this labeling information being attached to the inner nodes in the sense that a bit (called *flip*) is associated with each inner node that tells, when defined, which child is to the left and which is to the right.

Given a two-tree $(T_1, T_2; M)$ with labelings λ_i, our algorithm will basically branch on all possible settings of p to either L or R on the attachment links $\mathfrak{A}(G_i)$ (that do not contradict earlier settings $\neq *$) for the subgraphs $G_i = T_i\langle\lambda_i^{-1}(Q)\rangle$ of T_i for all possible contradicting quadruples Q. Observe that whenever a parent link of some node is assigned L, then the parent link of its sibling will be assigned R for reasons of consistency (and vice versa). After having considered all possible combinations, we can safely remove G_i from T_i: since also the attachment links have been fixed in their positions, it is clear how to finally produce a drawing, given a part of the original two-tree that can be drawn without crossings, plus giving information (obtained by complete branching) where to place all remaining links.

The problem we are facing is that we have to ensure that the natural parameter of this problem, i.e., the given budget k of tolerable crossings, is decremented in each branching step. So, our strategy will be to only branch at contradicting structures if this gives us a gain in each branch. To simplify matters, we assume that only those leaves that participated in

Algorithm 3 Procedure embed-TTCM: embed a two-tree with at most k crossings.

Require: A two-tree $(T_1, T_2; M)$; permanent link information p; two orderings $\prec_1$ and $\prec_2$ of the leaves; a parameter k.
Ensure: YES iff $(T_1, T_2; M)$ can be drawn with $\leqslant k$ crossings so that p, $\prec_1$, $\prec_2$ are respected.

if $k < 0$ **then**
  return NO
else if $k = 0$ **then**
  return embed-TT$''(T_1, T_2, M, p, \prec_1, \prec_2)$
5: {for embed-TT, see to Algorithm 1; it has to be, in addition, verified by this variant that the leaf orderings are respected; this might influence in particular the otherwise arbitrary choice of the flip at the root.}
else
  if there is a small contradicting structure S **then**
    branch on all possible flips for $\mathfrak{A}(T_i\langle S\rangle)$ with recursive calls on embed-TTCM, where S is deleted from the new instances and M, p are accordingly modified. In particular, the leaf orderings $\prec_1$ and $\prec_2$ have to be further fixed by transitivity.
  else
10: return embed-TT$'(T_1, T_2, M, p, \prec_1, \prec_2, k)$
    {This is another variant of Algorithm 1 primarily used to find new contradicting small structures as described in the proof; k is only needed if embed-TT$'$ recursively calls embed-TTCM.}
  end if
end if

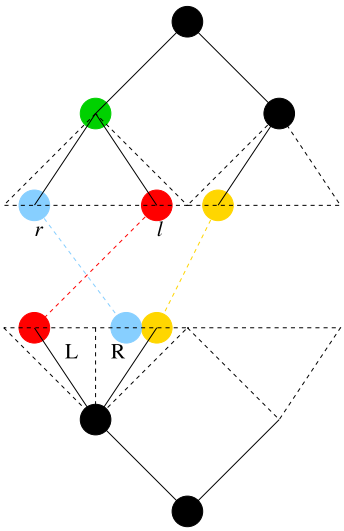

Fig. 7. A further contradiction for embed-TT$'$.

those contradicting structures we earlier branched on have been accounted for in the branching process. This is achieved by deleting contradicting structures that have been accounted for from the instance (together with their attachment links).

Extending our previous notions, as *contradicting structures*, we will view contradicting quadruples (as before) and contradicting pairs, i.e., pairs of labels a, b where say the upper tree fixes $a \prec_1 b$ and the lower tree fixes $b \prec_2 a$ (due to the flips that are fixed in both trees). Notice that this may happen even if we remove contradictory structures after branching: namely, this branching will implicitly ensure more and more of the leaf orderings $\prec_1$ and $\prec_2$. Moreover, since these orderings are to be transitive, further parts of these relations will be fixed by transitivity, and these transitive fixes will remain even if we later remove the very reason for fixing these orders.

Theorem 6. TWO-TREE CROSSING MINIMIZATION *is in* $\mathcal{FPT}$. *More precisely, the problem can be solved in time* $\mathcal{O}^*(c^k)$ *for some constant* c.

Proof. A pseudo-code sketching our proposed procedure is sketched in Algorithm 3. To fully understand Algorithm 3, we still have to explain what to do when there are no longer contradicting quadruples or pairs to be found (line 9): we will then start a procedure very similar to embed-TT. The only difference is that it will not find any of the contradictions described in the proof of Theorem 4 (since there are no contradicting quadruples). Now, a temporary labeling could contradict the already established permanent labeling. In that case, the permanent labeling would already exist upon first calling embed-TT$'$, so that we face the situation depicted in Fig. 7. This figure is understood as follows: the green inner node indicates a flip that has been determined. As the labels L and R indicate, this flip would go the other way round according to the temporary link information propagation. The permanent labeling must have a reason (otherwise, we could just interchange the meaning of L and R): namely, there must be a third leaf (the yellow one) that is residing somewhere in the "right branch" of the upper tree but in the "left branch" in the bottom tree (on a previous level of recursion). Hence, we also get a

crossing if we draw the left part of bottom tree coherently with the flip at the green node. Upon finding such an erroneous situation, we would consider the depicted leaves labeled red, blue and yellow as a contradicting structure and branch on all attachment points as before, recursively calling embed-TTCM again. □

Unfortunately, the constant c is still quite huge (about 2^8) due to the many attachment links whose combinations must be tested. Namely, at most flips of three inner nodes pertaining to a quadruple tree plus one further flip inn a parent node has to be tested in each tree. We can see our result therefore as a preliminary one, mainly providing a classification of the problem. However, in practice we would assume that the branching is indeed much better for several reasons:

- The sketched bad situation will be found only very rarely at the very beginning; later, it is to be expected that some permanent information and also some leaf orderings are already set in a way that the number of possible branches is largely restricted.
- In many branches, the number of crossings is reduced by more than one; this will largely improve on the involved recursions.
- We did not see into the "real crossing numbers" that are incurred by fixing several flips; in an implementation, one would better count the number of crossings induced by a certain permanent information fix in the original two-tree: the number of crossings found this way will be greater in general than our quite cautious estimate. Conversely, this means that we can further lower the budget k in most of the branches, hence further largely improving on our run times.

6. Conclusion

We have considered two-layer crossing minimization problems for the purpose of comparing two unordered trees. We derived $\mathcal{NP}$-completeness results and efficient polynomial-time algorithms as well as $\mathcal{FPT}$-algorithms. The following open problems are worthwhile to consider:

- Determine the (parameterized) complexity of the maximum planar submatching variant TTDE and prove $\mathcal{NP}$-completeness. Such a result is (possibly surprisingly) still missing also for the variant OTDE, where the order of one tree is fixed. This issue is also discussed in a Dagstuhl meeting on Graph Drawing in 2005, see the corresponding Open Problems Report as available from: drops.dagstuhl.de/opus/volltexte/2006/338/pdf/05191.SWM2.Paper.338.pdf.
- Extend the search-tree techniques to d-ary trees. From the first sight, an additional factor of $d!$ shows up. This problem has clustering applications, see [2]. More generally, in [35], the case of arbitrary (non-fixed) degree has been considered. This problem immediately reduces to a simple two-sided crossing-minimization problem, where even the one-side-fixed case is $\mathcal{NP}$-complete, see [18]. Note that different (and much more costly) $\mathcal{FPT}$-techniques for solving these problems were proposed in [35].
- Consider the weighted version of TTCM, where the crossings have higher weights if they occur between edges of larger different subtrees. This models a penalty on strongly misclassifying species: in one tree, two species might appear to be closely related, while in another tree, they appear to be only loosely related.
- The connections to 4-HITTING SET we exhibited for TTDE provides a factor-4 approximation; we do not see any constant-factor approximation for TTCM. Moreover, it might be possible to give more efficient direct parameterized and approximation algorithms for TTDE that do not use the detour via 4-HS. Similar comments apply to OTDE and 3-HS.
- Tree problems arising from phylogenetics occasionally display combinatorial reductions to small trees (like quadruple trees in our case); another example is the MINIMUM QUARTET INCONSISTENCY PROBLEM, where triple trees are important, see [7,22]. It would be interesting to investigate other tree problems from this combinatorial perspective. From an algorithmic perspective, (parameterized) complexity investigations is a natural research topic.

Acknowledgments

Thanks to G. Liotta and D. Huson for motivation and discussions, and to S. Näher for the pointer to the level-linked (2–4) trees.

References

[1] M.S. Bansal, W.-C. Chang, O. Eulenstein, D. Fernández-Baca, Generalized binary tanglegrams: Algorithms and applications, in: S. Rajasekaran (Ed.), Bioinformatics and Computational Biology BICoB, in: Lecture Notes in Comput. Sci., vol. 5462, Springer, 2009, pp. 114–125.
[2] Z. Bar-Joseph, D. Gifford, T. Jaakkola, Fast optimal leaf ordering for hierarchical clustering, Bioinformatics 17 (2001) 22–29.
[3] W. Barth, M. Jünger, P. Mutzel, Simple and efficient bilayer cross counting, J. Graph Algorithms Appl. 8 (2004) 179–194.
[4] F. Baumann, C. Buchheim, F. Liers, Exact crossing minimization in general tanglegrams, Technical Report zaik2009-581, Zentrum für Angewandte Informatik Köln, LehrstuhlJünger, March 2009.
[5] P. Bertolazzi, G. Di Battista, G. Liotta, C. Mannino, Optimal upward planarity testing of single-source digraphs, SIAM J. Comput. 27 (1998) 132–169.
[6] P. Bille, A survey on tree edit distance and related problems, Theoret. Comput. Sci. 337 (2005) 217–239.
[7] D. Bryant, M. Steel, Constructing optimal trees from quartets, J. Algorithms 38 (1) (2001) 237–259.

[8] K. Buchin, M. Buchin, J. Byrka, M. Nöllenburg, Y. Okamoto, R.I. Silveira, A. Wolff, Drawing (complete) binary tanglegrams, in: I.G. Tollis, M. Patrignani (Eds.), Graph Drawing, 16th International Symposium, GD 2008, in: Lecture Notes in Comput. Sci., vol. 5417, Springer, 2009, pp. 324–335.
[9] M.A. Charleston, Principles of cophylogenetic maps, in: Lecture Notes in Phys., vol. 585, Springer, 2002, pp. 122–147.
[10] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice Hall, 1999.
[11] R.G. Downey, M.R. Fellows, Parameterized, Complexity, Springer, 1999.
[12] V. Dujmović, M.R. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F.A. Rosamond, M. Suderman, S. Whitesides, D.R. Wood, On the parameterized complexity of layered graph drawing, in: F. Meyer auf der Heide (Ed.), 9th Annual European Symposium on Algorithms ESA, in: Lecture Notes in Comput. Sci., vol. 2161, Springer, 2001, pp. 488–499.
[13] V. Dujmović, M.R. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F.A. Rosamond, M. Suderman, S. Whitesides, D.R. Wood, A fixed-parameter approach to 2-layer planarization, Algorithmica 45 (2006) 159–182.
[14] V. Dujmović, H. Fernau, M. Kaufmann, Fixed parameter algorithms for one-sided crossing minimization revisited, J. Discrete Algorithms 6 (2008) 313–323.
[15] V. Dujmović, S. Whitesides, An efficient fixed parameter tractable algorithm for 1-sided crossing minimization, Algorithmica 40 (2004) 15–32.
[16] T. Dwyer, F. Schreiber, Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation, in: Proc. Australasian Symp. on Information Visualisation (InVis.au 2004), in: CRPIT, vol. 35, 2004, pp. 109–115.
[17] P. Eades, S. Whitesides, Drawing graphs in two layers, Theoret. Comput. Sci. 13 (1994) 361–374.
[18] P. Eades, N. Wormald, Edge crossings in drawings of bipartite graphs, Algorithmica 10 (1994) 379–403.
[19] M.R. Fellows, The Robertson–Seymour theorems: A survey of applications, Contemp. Math. 89 (1989) 1–18.
[20] H. Fernau, Parameterized Algorithmics: A Graph-Theoretic Approach, Habilitationsschrift, Universität Tübingen, Germany, 2005, the mentioned results on 4-HS have been accepted for publication to International Journal of Computer Mathematics.
[21] H. Fernau, Two-layer planarization: Improving on parameterized algorithmics, J. Graph Algorithms Appl. 9 (2005) 205–238.
[22] J. Gramm, R. Niedermeier, Quartet inconsistency is fixed parameter tractable, in: A. Amir, G.M. Landau (Eds.), Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001), in: Lecture Notes in Comput. Sci., vol. 2089, Springer, 2001, pp. 241–256.
[23] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.
[24] D. Huson, private communication, 2005.
[25] T. Jiang, L. Wang, K. Zhang, Alignment of trees—an alternative to tree edit, Theoret. Comput. Sci. 143 (1995) 137–148.
[26] P. Kilpeläinen, H. Mannila, Ordered and unordered tree inclusion, SIAM J. Comput. 24 (1995) 340–356.
[27] P. Klein, S. Tirthapura, D. Sharvit, B. Kimia, A tree-edit-distance algorithm for comparing simple, closed shapes, in: Proc. of 11th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2000, pp. 696–704.
[28] A. Lozano, R.Y. Pinter, O. Rokhlenko, G. Valiente, M. Ziv-Ukelson, Seeded tree alignment, IEEE/ACM Trans. Comput. Biol. Bioinformatics 5 (4) (2008) 503–513.
[29] K. Mehlhorn, Data Structures and Algorithms 1: Sorting and Searching, Monogr. Theoret. Comput. Sci. EATCS Ser., Springer, 1984.
[30] M. Nöllenburg, M. Völker, A. Wolff, D. Holten, Drawing binary tanglegrams: An experimental evaluation, in: I. Finocchi, J. Hershberger (Eds.), Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX 2009, SIAM, 2009, pp. 106–119.
[31] R.D.M. Page (Ed.), Tangled Trees: Phylogeny, Cospeciation, and Coevolution, University of Chicago Press, 2002.
[32] R. Ramesh, I. Ramakrishnan, Nonlinear pattern matching in trees, J. ACM 39 (1992) 295–316.
[33] C.L. Schardl, K.D. Craven, A. Lindstrom, S. Speakman, A. Stromberg, R. Yoshida, A novel test for host-symbiont codivergence indicates ancient origin of fungal endophytes in grasses, arXiv:q-bio/0611084v2 [q-bio.PE], 6 October 2007.
[34] M. Suderman, Layered Graph Drawing, PhD thesis, McGill University, Montréal, 2005.
[35] B. Venkatachalam, J. Apple, K.St. John, D. Gusfield, Untangling tanglegrams: Comparing trees by their drawings, in: I.I. Mandoiu, G. Narasimhan, Y. Zhang (Eds.), Bioinformatics Research and Applications, 5th International Symposium, ISBRA, in: Lecture Notes in Comput. Sci., vol. 5542, Springer, 2009, pp. 88–99.
[36] M. Wahlström, Algorithms, measures and upper bounds for satisfiability and related problems, PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden, 2007.
[37] K. Zhang, D. Shasha, Simple fast algorithms for the edit distance between trees and related problems, SIAM J. Comput. 18 (1989) 1245–1262.